K-D Trees Are Better when Cut on the Longest Side

Matthew Dickerson[1], Christian A. Duncan[2], and Michael T. Goodrich[3] *

[1] Dept. of Math & Comp. Sci., Middlebury College, Middlebury, VT 05753
dickerso@middlebury.edu
[2] Algorithms & Complexity Group, Max-Planck-Inst. für Informatik, D-66123 Saarbrücken, Germany duncan@mpi-sb.mpg.de
[3] Dept. of Comp. Sci., Johns Hopkins Univ., Baltimore, MD 21218
goodrich@jhu.edu

Abstract. We show that a popular variant of the well known k-d tree data structure satisfies an important packing lemma. This variant is a binary spatial partitioning tree T defined on a set of n points in $\mathbb{R}^d$, for fixed $d \geq 1$, using the simple rule of splitting each node's hyperrectangular region with a hyperplane that cuts the longest side. An interesting consequence of the packing lemma is that standard algorithms for performing approximate nearest-neighbor searching or range searching queries visit at most $O(\log^{d-1} n)$ nodes of such a tree T in the worst case. Traditionally, many variants of k-d trees have been empirically shown to exhibit polylogarithmic performance, and under certain restrictions in the data distribution some theoretical expected case results have been proven. This result, however, is the first one proving a worst-case polylogarithmic time bound for approximate geometric queries using the simple k-d tree data structure.

1 Introduction

The humorist Erma Bombeck is quoted as once praying,

> "Lord, if you can't make me thin, then make my friends look fat."

In computational geometry, however, we desire objects that are fat, not thin. That is, we desire objects that have a bounded aspect ratio, and this desire is particularly true in the context of spatial partitioning data structures. However it turns out occasional skinny objects are acceptable as long as there are not too many of them and their neighbors are not skinny. Indeed, a recent paper on this topic by Maneewongvatana and Mount [14] can be viewed as turning the Bombeck quote around by stating,

> "It's okay to be skinny, if your friends are fat."

* This research partially supported by NSF grant CCR-9732300 and ARO grant DAAH04-96-1-0013.

In the same way, we are interested in studying spatial partitioning data structures that allow some regions to be skinny so long as the regions around them tend to be fat. We formalize this into a framework we call the *quasi-balanced aspect ratio* tree framework, or *quasi-BAR* tree, for short. We show that trees that fall into this class satisfy an important packing lemma that implies they can be used efficiently for approximate nearest-neighbor and range searching queries. The major result of this paper is to show that a popular variant of the well-known k-d tree data structure falls into this framework. This fact implies that several standard k-d tree searching algorithms actually run in polylogarithmic time when applied to this k-d tree variant.

1.1 Prior Related Work

Bentley [3] introduced the k-d tree data structure for storing a set S of n points in $\mathbb{R}^d$, for fixed constant $d \geq 1$.[1] The idea behind this structure is quite simple. We recursively assume that we have a bounding box, B, that contains all the points in S. We choose one of the d coordinate directions and find the median point in S with respect to this direction. We then cut the box B by a hyperplane perpendicular to this direction so as to go through this median point, and we recurse on the two regions and point sets that this cutting determines. Thus, by repeating this operation until the number of points inside the bounding box is less than some constant, we define a binary spatial partitioning (BSP) tree structure that has $O(\log n)$ depth. Variants of the k-d tree structure are distinguished by the heuristic applied to determine the cut directions. In the original paper by Bentley [3], the heuristic was to simply alternate between the d possible directions in a round-robin fashion.

Friedman, Bentley, and Finkel [12] study an alternate definition, where the cut is defined perpendicular to the direction with *maximum spread*, that is, the direction where the difference between coordinate values in that direction in S is largest. They show that for data distributions with bounded density, k-d trees defined using this heuristic can achieve $O(\log n)$ expected query times for approximate nearest-neighbor and range searching. Silva-Filho [17] studies the choosing of a cutting hyperplane based on probabilistic considerations. Sproull [18] considers several other heuristics for k-d trees in practice. One standard simplification is to use the following splitting rule:

> **The Longest Side Rule:** Choose a splitting hyperplane perpendicular to the longest side of the bounding box B.

This heuristic is often used in practice, since it is so simple to implement and tends to mimic the behavior of the maximum-spread heuristic.

Bentley [4] reports on experimental results for k-d trees defined using the maximum-spread heuristic, showing that for a variety of input distributions this variation performs remarkably well for nearest-neighbor and other searches. We are not familiar with any previous work that reports a non-trivial worst-case

[1] We assume the dimension d is fixed throughout this paper.

upper bound for using a k-d tree for nearest-neighbor searching, however, or even approximate nearest-neighbor searching. Still, for range searching queries, where one wishes to count (or report) the points inside a given axis-parallel hyper-rectangle, Lee and Wong [13] show that the standard k-d tree structure can be used to achieve a worst-case query time of $\Theta(n^{(d-1)/d})$ (plus output size in the reporting case). Silva-Filho [16] shows that this bound even holds in the average case for range searching in the standard k-d tree structure.

Deviating from the strict k-d tree approach, there have been several data structures developed for efficiently performing approximate nearest-neighbor searching and range search queries. For example, fair-split trees [5], defined by Callahan and Kosaraju, achieve logarithmic query time for approximate nearest-neighbor searching, although the trees they define do not necessarily have $O(\log n)$ depth. The balanced box-decomposition (BBD) trees of Arya et al. [2, 1], on the other hand, have $O(\log n)$ depth and have regions with good aspect ratio, and achieve logarithmic-time performance for approximate nearest-neighbor and range searching. The BBD tree deviates from the k-d tree approach by introducing holes into the middle of regions. The balanced-aspect ratio (BAR) trees of Duncan, Goodrich, and Kobourov [9, 10] achieve similar bounds to those of Arya et al., but do so using simple hyperplane cuts at each internal node. However, their BSP trees are not strictly k-d trees as the cuts need not be axis-aligned and do not always equally subdivide the set of points in a region.

Previous work most similar to ours is a recent paper by Maneewongvatana and Mount [14], which provides a packing lemma analogous to one we derive. Their lemma is defined for a specific k-d tree splitting rule, called the *sliding-midpoint split*, that allows the resulting k-d tree to have $\Theta(n)$ depth in the worst case. Thus, it does not seem possible to use this tree to achieve polylogarithmic query times in the worst case for approximate nearest-neighbor and range searching queries. In addition, the proof technique used by Maneewongvatana and Mount is quite different from the one of this paper.

1.2 Our Results

In this paper, we present a general framework for proving that families of BSP trees have good worst-case performance for approximate nearest-neighbor and range searching queries. The framework is based on a relaxed form of the BAR tree of Duncan, Goodrich, and Kobourov [9, 10], which we call the *quasi-BAR tree*. The most important component of our approach is that BSP trees falling into our framework satisfy an important packing lemma, which implies efficient performance of several approximate query operations. Since these structures are straight-forward partitioning trees, we only need linear space to store them as well.

In Section 3, we show that a k-d tree defined using the *longest-side* splitting rule falls into our quasi-BAR tree framework. Thus, longest-side k-d trees consequently achieve polylogarithmic worst-case performance for approximating geometric queries. Although these bounds are certainly not better than the logarithmic worst-case bounds of fair-split trees [5], BBD trees [2, 1], and BAR

trees [9, 10], they are nonetheless intriguing for a number of reasons. For example, many empirical results (e.g., see [4, 8, 12]) show that k-d trees defined with the maximum-spread and longest-side splitting rules perform well in practice, but no worst-case theoretical evidence has been given to support these observations. Our analysis shows that k-d trees defined by the longest-side splitting rule require that we visit at most $O(\log^d n)$ nodes in the worst case to answer any approximate nearest-neighbor and range searching queries on a set of n points in $\mathbb{R}^d$, for any fixed constant $d \geq 1$. We also show that such k-d trees perform exact orthogonal range searches in the plane quite efficiently. We discuss the main ideas behind these results in the sections that follow.

2 A Framework for BSP Trees with Packing Lemmas

Prior work indicates that the performances of BBD [2, 1] and BAR [9, 10] trees are rooted mainly in the bounds from important packing lemmas. This property is due to the manner in which standard BSP searching algorithms proceed down a search tree to answer a given query. In this section we show how to extend these previous specific packing lemmas into a framework for establishing packing lemmas for other types of BSP trees. Thus, for the sake of completeness, let us review some BSP search algorithms.

2.1 A Standard BSP Nearest-Neighbor Searching Algorithm

Let S be a set of n points in $\mathbb{R}^d$, and let T be a BSP tree defined on S. We use $\delta(p, q)$ to denote the distance between a point p and a point q in $\mathbb{R}^d$. We often refer to a node u in a given tree with an associated region R_u. In these cases, for convenience we use $\delta(u, q) = \delta(R_u, q)$, where $\delta(R_u, q)$ is the distance from q to the region R_u.

Definition 2.1. *For a set S of points in $\mathbb{R}^d$, a query point $q \in \mathbb{R}^d$, and $\epsilon > 0$, a point $p \in S$ is a* $(1+\epsilon)$-nearest neighbor of q *if $\delta(p, q) \leq (1+\epsilon)\delta(p^*, q)$, where p^* is the true nearest neighbor to q.*

In other words, such a p is within a constant error factor of the true nearest neighbor. Given a query point p and an error parameter $\epsilon > 0$ we can use the following algorithm, which is similar to an algorithm of Arya et al. [2], to find an approximate nearest-neighbor to p in S.

We initialize a priority queue Q with the root node of T. Let p be the current nearest neighbor identified during our search, initially some point at ∞. At every stage, extract from Q the node u that is the nearest to q. If $(1+\epsilon)\delta(u, q) \geq \delta(p, q)$, we exit and return p as the $(1+\epsilon)$-approximate nearest neighbor. The following operations are repeated until the next node is extracted from Q. If u is not a leaf, let u_1 and u_2 be u's children. Without loss of generality, let u_1 be the node nearer to q, i.e. $\delta(u_1, q) \leq \delta(u_2, q)$. We insert u_2 onto the queue and continue with u_1, bypassing the extraction process. If u is a leaf node, we let $S' = S \cap u$ be the (constant-size) set of data points in u. For all $p' \in S'$, if $\delta(p', q) < \delta(p, q)$, we let $p \leftarrow p'$. We continue by extracting the next nearest node from Q.

Definition 2.2. *For a set S of points in $\mathbb{R}^d$, a query point $q \in \mathbb{R}^d$, and $\epsilon > 0$, a point $p \in S$ is a* $(1-\epsilon)$-farthest neighbor of q *if $\delta(p,q) \leq \delta(p^*,q) - \epsilon D$, where p^* is the true farthest neighbor to q and D is the diameter of the point set.*

Definition 2.3. *For a set S of points in $\mathbb{R}^d$, a query region Q with diameter O_Q, and $\epsilon > 0$, an* ϵ-approximate range query *returns (or counts) a set S' such that $S \cap Q \subseteq S' \subseteq S$ and for every point $p \in S'$, $\delta(p,Q) \leq \epsilon O_Q$.*

In the full version, we also review the standard BSP algorithms for finding approximate farthest neighbors and approximate range searching.

2.2 Quasi-BAR Trees

A foundational construct in our quasi-BAR tree framework is the need to bound non-trivially the number of regions *piercing* a set we call a region annulus.

Definition 2.4. *For any region R, we define a* region annulus with radius r *$A_{R,r}$ to be the set of all real points $p \in \mathbb{R}^d$ such that $p \notin R$ and $\delta(p,R) < r$. A region R'* pierces *$A_{R,r}$ if and only if there exists two real points $q_1, q_2 \in R'$ such that $q_1 \in R$ and $q_2 \notin R \cup A_{R,r}$.*

Basically, a region annulus is the set of points outside but near the border of R. If R were a spherical region with radius r', this would be the traditional notion of an annulus with radii r' and $r' + r$, respectively. A region R' pierces this annulus if it lies partially inside R and partially farther than r away.

Definition 2.5. *Given any region annulus A and BSP tree T, let $\mathcal{P}_T(A)$ denote the largest set of disjoint nodes in T whose associated regions pierce the region annulus A. A class of binary space partitioning trees is a* $\rho(n)$-quasi-BAR tree *if, for any tree T in the class constructed on a set S of n points in $\mathbb{R}^d$ and any region annulus A, $|\mathcal{P}_T(A)| \leq \rho(n) V_A / r^d$, where V_A and r are the volume and associated radius (intuitively, the "width") of A, respectively (see Definition 2.4).*

In other words, the number of nodes with regions intersecting an annulus A in a quasi-BAR tree defined on a set of n points is bounded by a function of n times the "relative thickness" of A. The advantage of the quasi-BAR tree definition is that it allows us to prove the following theorems.

Theorem 2.6. *Suppose we are given a $\rho(n)$-quasi-BAR tree T with depth $D_T = \Omega(\log n)$ constructed on a set S of n points in $\mathbb{R}^d$. For any query point q, the standard search algorithms find respectively a $(1+\epsilon)$-nearest and a $(1-\epsilon)$-farthest neighbor to q in $O(\epsilon^{1-d}\rho(n)D_T)$ time.*

Theorem 2.7. *Suppose we are given a $\rho(n)$-quasi-BAR tree T with depth D_T constructed on a set S of n points in $\mathbb{R}^d$. For any convex query region Q, one can perform a counting (or reporting) ϵ-approximate range searching query in T in $O(\epsilon^{1-d}\rho(n)D_T)$ time (plus output size in the reporting case). For any general non-convex query region Q, the time required is $O(\epsilon^{-d}\rho(n)D_T)$ (plus output size).*

The proofs for these theorems follow directly from proofs in Arya et al. [2] and Duncan et al. [10]. The main concept is in proving that all visited "leaf" nodes pierce a region annulus A such that $V_A/r^d = O(\epsilon^{1-d})$. We leave the proof for the full version of the paper.

So long as a class of trees satisfies a packing lemma, we can allow for skinny regions and still achieve good worst-case performance in several approximate geometric queries. The important feature in such cases is to minimize the packing function $\rho(n)$. The difficulty is in developing a data structure that actually guarantees a non-trivial packing function. Surprisingly, such a structure already exists and is a commonly used k-d tree variant.

3 Longest-side K-d Trees

> "... just don't have too many other skinny friends."

We begin our proof of the existence of a non-trivial quasi-BAR tree by reviewing the definition of the longest-side k-d tree. We show that these regions, which may be skinny, do not have many skinny regions nearby of comparable size.

Definition 3.1. *The* longest-side k-d tree *is the tree constructed by recursively dividing the point set associated with each node u in half by cutting perpendicular to the longest axis-orthogonal side of R_u.*

The establishment of a non-trivial packing function $\rho(n)$ for the longest-side k-d trees depends upon the following *Hypercube Stabbing Lemma*, whose proof technique may potentially be of use in analyzing other BSP trees.

Lemma 3.2 (Hypercube Stabbing Lemma). *Suppose we are given a longest-side k-d tree T constructed on a set S of n points in $\mathbb{R}^d$. Let L be the largest set of disjoint nodes in T such that every region in L intersects at least two opposing sides of a hypercube H. Then $|L|$ is $O(\log^{d-1} n)$.*

Proof. Let h be the side length of H. Let us assume, wlog, that the hypercube H is fully contained in the region R associated with the root of T. To generate the largest set L of disjoint nodes in T intersecting at least two opposing sides of H, we first observe that if a node intersects a side of H then all of its ancestors must also intersect this side. Therefore, L consists of all nodes in T none of whose children intersect two opposing sides of H. We now show that $|L|$ is $O(2^d \log^{d-1} n)$.

For this analysis, let us classify a node u in T and its associated region R_u by the number of sides of H that R_u intersects. In particular, let the class $\mathcal{C}(i,j)$ denote the set of regions in T such that each region R intersects i pair of opposing sides of H and j *other* sides (that is, without their "partners" in the same direction). Notice that $i + j \leq d$. More importantly, for any region in L belonging to class $\mathcal{C}(i,j)$ we know $i \geq 1$ (by the definition of L). Also, note that the root of T is in the class $\mathcal{C}(d,0)$.

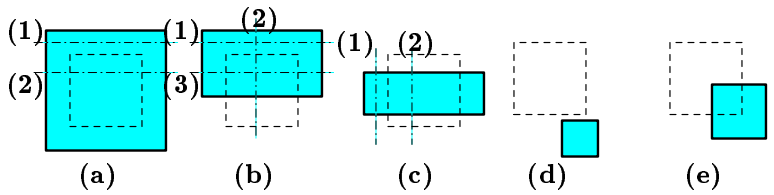

Fig. 1. The various possibilities for the class $\mathcal{C}$ in the plane. The hypercube H is shown in outline while a region in T is shown shaded. We label the various cuts by the classes of child regions they produce. Examples of classes (a) $\mathcal{C}(2,0)$, (b) $\mathcal{C}(1,1)$, and (c) $\mathcal{C}(1,0)$. Regions (d) and (e) cannot be in L because they do not intersect two opposing sides

Let R, associated with some internal node v in T, be a region belonging to class $\mathcal{C}(i,j)$. Since we are only concerned with regions intersecting at least one pair of opposing sides of H, we look at those cases where $i \geq 1$. For the cut c that divides R into two subregions R_1 and R_2, consider the possible cases of classes to which the two children belong.

1. The cut c may be entirely outside of H. In this case, one of v's child regions, say R_2, lies entirely outside of H. Hence, $R_2 \in \mathcal{C}(0,0)$ and $R_1 \in \mathcal{C}(i,j)$.
2. The cut c intersects the inside of H and is perpendicular to a dimension in which the region R spans H on both sides. In this case, both R_1 and R_2 intersect H but with $i-1$ opposing pairs and $j+1$ other sides. Therefore, $R_1, R_2 \in \mathcal{C}(i-1, j+1)$.
3. The cut c intersects the inside of H and is perpendicular to a dimension in which the region R intersects H on just one side. In this case, both R_1 and R_2 intersect H. Moreover, both R_1 and R_2 have the same number of dimensions in which they span H on both sides. One of these regions, however, has one fewer dimension in which it spans H on a single side. That is, without loss of generality, $R_1 \in \mathcal{C}(i,j)$ and $R_2 \in \mathcal{C}(i,j-1)$.

We argue that these are all the possible cases. Case 1 includes all cuts that do not intersect H. If the cut intersects the inside of H, then it is perpendicular to some dimension, k. Along the dimension k, the region R must intersect 2 sides of H, 1 side of H, or 0 sides of H. The first two of these possibilities are covered by Cases 2 and 3 above. The last possibility cannot occur, however, because of the longest-side splitting rule. In particular, if a region R intersects two opposing sides of H, then the longest side of R is at least h, but the side of R in dimension k must have length less than h.

We can now proceed to our evaluation by utilizing a recurrence relation. Let u be a node in T whose associated region R is in $\mathcal{C}(i,j)$. We define $c(u,i,j)$ to be the largest number of disjoint regions associated with descendents of u which intersect at least one pair of opposing sides of H. In other words, every region belongs to a class $\mathcal{C}(i',j')$ for some $i' \geq 1$. Let u_1 and u_2 be u's two children.

The recurrence follows directly from the three cases above:

$$c(u,i,j) = \max \begin{cases} c(u_1,i,j) \\ c(u_1,i-1,j+1) + c(u_2,i-1,j+1) \\ c(u_1,i,j) + c(u_2,i,j-1) \end{cases}$$

Our base cases occur either when u is a leaf l_u (u_1 and u_2 do not exist) or when $i = 0$. In the former $c(l_u,i,j) = 1$. In the latter $c(u,0,j) = 0$ by definition.

We use induction to show that $c(u,i,j) \leq 2^i(\log|u|)^{i+j-1}$, where $|u|$ is the number of points in the subtree rooted at u. For convenience, we maintain that every leaf node has at least two points in it. This simply implies that $\log|u| \geq 1$. We also assume that $|u|$ is an even number, therefore, $|u_1| = |u_2| = |u|/2$. The base cases follow from brute force. Our analysis relies on two fundamental inequalities for $a \geq 1$:

$$\begin{array}{ll} a^b \geq (a-1)^b & \text{For } b \geq 0 \\ a^b \geq (a-1)^b + (a-1)^{b-1} & \text{For } b \geq 1 \end{array}$$

We show that the inductive case holds for any of the three possible recurrences. If u is not a leaf node, $\log|u| \geq 1$, and since $i \geq 1$, we know that $i + j - 1 \geq 0$.
Case 1. Let $c(u,i,j) = c(u_1,i,j)$. Using our inductive hypothesis, we have

$$\begin{aligned} c(u,i,j) &= c(u_1,i,j) \\ &\leq 2^i(\log|u_1|)^{i+j-1} \\ &= 2^i(\log|u|-1)^{i+j-1} \\ &\leq 2^i(\log|u|)^{i+j-1}. \end{aligned}$$

Case 2. Let $c(u,i,j) = c(u_1,i-1,j+1) + c(u_2,i-1,j+1)$. By induction,

$$\begin{aligned} c(u,i,j) &= c(u_1,i-1,j+1) + c(u_2,i-1,j+1) \\ &\leq 2^{i-1}(\log|u_1|)^{i+j-1} + 2^{i-1}(\log|u_2|)^{i+j-1} \\ &= 2^i(\log|u|-1)^{i+j-1} \\ &\leq 2^i(\log|u|)^{i+j-1}. \end{aligned}$$

Case 3. Let $c(u,i,j) = c(u_1,i,j) + c(u_2,i,j-1)$. In this case, recall that the cut lies in a direction in which the region R_u intersects H on only one side. This implies that $j \geq 1$ and thus that $i + j - 1 \geq 1$. Again by induction,

$$\begin{aligned} c(u,i,j) &= c(u_1,i,j) + c(u_2,i,j-1) \\ &\leq 2^i(\log|u_1|)^{i+j-1} + 2^i(\log|u_2|)^{i+j-2} \\ &= 2^i(\log|u|-1)^{i+j-1} + 2^i(\log|u|-1)^{i+j-2} \\ &\leq 2^i(\log|u|)^{i+j-1}. \end{aligned}$$

The solution to the recurrence relation is now proven. We only need to recall that the region associated with the root of the tree T spans H on both sides in every dimension. Thus, $|L| \leq c(T,d,0) \leq 2^d(\log n)^{d-1}$. □

This Hypercube Stabbing Lemma might at first seem to be unrelated to the definition of a quasi-BAR tree, which depends heavily on the notion of a region annulus, but this is not the case, as we show in the following theorem.

Theorem 3.3. *Suppose we are given a longest-side k-d tree T constructed on a set S of n points in $\mathbb{R}^d$. Then the packing function $\rho(n)$ of T for a region annulus A is $O(\log^{d-1} n)$. That is, the class of longest-side k-d trees is an $O(\log^{d-1} n)$-quasi-BAR tree.*

Proof. Let L be a set of disjoint regions from T piercing $A = A_{Q,r}$. Let $R \in L$ be any such region piercing A. Notice that $O_R \geq r/2$; therefore, we know that the longest side of the bounding box of R is certainly greater than $r/\sqrt{d}$. Let $\mathcal{H}$ be the smallest set of disjoint hypercubes with side length $r/(2\sqrt{d})$ that completely cover A. Notice that $|\mathcal{H}|$ is $O(V_A/r^d)$. Now, any region R that pierces A must intersect two opposing sides of at least one hypercube in $\mathcal{H}$ (see Figure 2). For any hypercube $H \in \mathcal{H}$, let $L' \subseteq L$ be the subset of regions in L that intersect two opposing sides of H. From the Hypercube Stabbing Lemma (3.2), we know that $|L'| = O(\log^{d-1} n)$. Therefore, we know that

$$|\mathcal{P}_T(A)| = |L| = |L'||\mathcal{H}| \leq c \log^{d-1} n V_A/r^d,$$

for some constant $c > 0$. From Definition 2.5, the class of longest-side k-d trees is therefore an $O(\log^{d-1} n)$-quasi-BAR tree. □

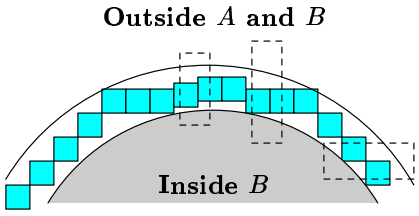

Fig. 2. An annulus A with associated region B. Any region in L, shown here with dashed lines, must cross two opposing sides of one of the hypercube boxes inside A

4 Applications

The theorems from previous sections immediately imply additional corollaries.

Corollary 4.1. *Suppose we are given a longest-side k-d tree T constructed on a set S of n points in $\mathbb{R}^d$. For any query point q, the standard algorithms for finding respectively a $(1+\epsilon)$-nearest or a $(1-\epsilon)$-farthest neighbor to q run in $O(\epsilon^{1-d} \log^d n)$ time in the worst case.*

Corollary 4.2. *Suppose we are given a longest-side k-d tree T constructed on a point set S in $\mathbb{R}^d$. For any convex query region, the standard algorithm for counting (or reporting) in an ϵ-approximate range searching query runs in $O(\epsilon^{1-d}\log^d n)$ wosrt-case time (plus output size in the reporting case). For any general query region, the worst-case time required is $O(\epsilon^{-d}\log^d n)$ (plus output size).*

4.1 Exact Range-Searching in the Plane

In $\mathbb{R}^2$, there is an additional interesting result that can also be proven for the longest-side k-d tree if we enhance each node's region to maintain the bounding box of the point set in the region, rather than just the region itself. We define the bounding box of a region to be the smallest axis-orthogonal box containing all points in the region. This assumption induces only a constant factor overhead and is also common in practice because of the time and space saved. Using this enhancement, we can actually show that any orthogonal range query in $\mathbb{R}^2$ can be answered *exactly* with a running time dependent only on the set size and the aspect ratio of the query. Thus, if the query region is fat, that is, has a low aspect ratio, then the running time becomes polylogarithmic.

Theorem 4.3. *Suppose we are given a bounding-box longest-side k-d tree T constructed on a set S of n points in $\mathbb{R}^2$. Let Q be any orthogonal query region with dimensions w_x and w_y and wlog let $w_x \geq w_y$. The standard range searching algorithm reports an* exact orthogonal range query *in $O((w_x/w_y)\log^2 n + k\log n)$ time, where k is number of points reported.*

Proof. Let us begin by breaking the region Q into $\alpha = \lceil w_x/w_y \rceil$ squares, labeled Q_i. Notice that each square has side length w_y (see Figure 3a). Let L be the set of all visited nodes which are neither trivially accepted nor rejected. Recall that a node is trivially accepted if it lies completely within the query region Q and trivially rejected if it does not intersect Q. The nodes in L are then exactly those nodes which do partially intersect Q. Let L' be the subset of L such that for any node $u \in L$ with child nodes u_l and u_r, $u \in L'$ if and only if $u_l, u_r \notin L$. Let $L'' \subseteq L'$ be the set of nodes in L' which intersect two opposing sides of Q. Furthermore, let $L_i \subseteq L''$ be the set of nodes in L'' which intersect two opposing sides of Q_i. From the Hypercube Stabbing Lemma (3.2), we know that $|L_i|$ is $O(\log n)$. Thus, $|L''|$ is $O(\alpha \log n)$.

Let us now look at the set $K = L' \setminus L''$. This is a disjoint set of nodes which intersect Q but do not intersect two opposing sides. First, notice that there can be at most 4 nodes in K which intersect two sides of Q, namely the four regions at the corners (see Figure 3b). We now use the bounding box enhancement. If the bounding box of a region R intersects only one side of Q, then there must exist a point p in R which lies inside of Q. Therefore, we know that $|K| \leq k + 4$.

If we combine our results, we see that

$$|L| \leq |L'| \log n = (|L''| + |K|) \log n \in O(\alpha \log^2 n + k \log n).$$

Consequently, the running time is $O(|L| + k) = O(\alpha \log^2 n + k \log n)$. □

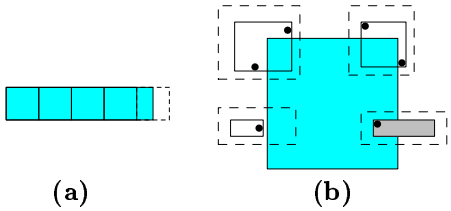

Fig. 3. (a) Breaking a region into a chain of squares. (b) A particular square showing potential region intersections *not* intersecting at least two opposing sides. The original regions prior to the bounding box are shown dashed. Notice only one type can intersect the square without being at a corner, and this intersection must contain a point of S

This result, although surprising, is still not better than the range searching data structure of Chazelle [6], which performs exact queries in the plane in $O(\log n)$ time and $O(n)$ space regardless of the aspect ratio of the query. It is simply interesting to note that such exact queries are achievable using a widely-used k-d tree data structure.

5 Conclusion

In this paper, we have developed a general framework for BSP trees that satisfy a packing lemma that qualifies them to be quasi-BAR trees. We also described some important applications that can be solved using this type of tree. In particular, we showed that the well-known longest-side k-d tree falls into this framework and is, therefore, capable of polylogarithmic time approximation queries. Although certainly not a theoretical improvement on the BBD tree structure [2, 1] or the BAR tree structure [9, 10], this result helps explain why longest-side k-d trees perform well in practice.

Several researchers [4, 15, 19] have studied the dynamic behavior of k-d trees, where items can be inserted and/or removed. A natural open question, then, is whether one can show if a natural dynamic variant of the k-d tree fits into our quasi-BAR tree framework and exhibits worst-case polylogarithmic update times and worst-case polylogarithmic query times for approximate nearest-neighbor and range searching. Another natural open question is whether any other variations of the k-d tree, such as maximum-spread, are $\rho(n)$-quasi-BAR trees for some polylogarithmic $\rho(n)$ function.

Acknowledgement

We would like to thank David Mount for several helpful conversations relating to the topics of this paper.

References

[1] S. Arya and D. M. Mount. Approximate range searching. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages 172–181, 1995.

[2] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching. In *Proc. 5th ACM-SIAM Sympos. Discrete Algorithms*, pages 573–582, 1994.

[3] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975.

[4] J. L. Bentley. K-d trees for semidynamic point sets. In *Proc. 6th Annu. ACM Sympos. Comput. Geom.*, pages 187–197, 1990.

[5] P. B. Callahan and S. R. Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *J. ACM*, 42:67–90, 1995.

[6] B. Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM J. Comput.*, 17:427–462, 1988.

[7] J. R. Driscoll, H. N. Gabow, R. Shrairaman, and R. E. Tarjan. Relaxed heaps: An alternative to Fibonacci heaps with applications to parallel computation. *Commun. ACM*, 31:1343–1354, 1988.

[8] C. A. Duncan. *Balanced Aspect Ratio Trees.* Ph.D. thesis, Dept. of Computer Science, Johns Hopkins Univ., 1999.

[9] C. A. Duncan, M. T. Goodrich, and S. G. Kobourov. Balanced aspect ratio trees and their use for drawing very large graphs. In *Proc. Graph Drawing '98, LNCS 1547*, pages 111–124. Springer-Verlag, 1998.

[10] C. A. Duncan, M. T. Goodrich, and S. G. Kobourov. Balanced aspect ratio trees: Combining the advantages of k-d trees and octrees. In *Proc. 10th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 300–309, 1999.

[11] M. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization problems. *J. ACM*, 34:596–615, 1987.

[12] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, 3:209–226, 1977.

[13] D. T. Lee and C. K. Wong. Worst-case analysis for region and partial region searches in multidimensional binary search trees and balanced quad trees. *Acta Inform.*, 9:23–29, 1977.

[14] S. Maneewongvatana and D. M. Mount. It's okay to be skinny, if your friends are fat. In *4th Annual CGC Workshop on Computational Geometry*, 1999.

[15] M. H. Overmars and J. van Leeuwen. Dynamic multi-dimensional data structures based on quad- and k-d trees. *Acta Inform.*, 17:267–285, 1982.

[16] Y. V. Silva Filho. Average case analysis of region search in balanced k-d trees. *Inform. Process. Lett.*, 8:219–223, 1979.

[17] Y. V. Silva Filho. Optimal choice of discriminators in a balanced k-d binary search tree. *Inform. Process. Lett.*, 13:67–70, 1981.

[18] R. F. Sproull. Refinements to nearest-neighbor searching. *Algorithmica*, 6:579–589, 1991.

[19] M. J. van Kreveld and M. H. Overmars. Divided k-d trees. *Algorithmica*, 6:840–858, 1991.